Digital Loggers

- About
- Products
- Support
- Quotes
- Applications

- Power Control
- Pro Switch
- Rack Mount PDU
- New 90-240V PDU
- 3-Phase PDU
- DC Power Switch
- DIN Power Relay
- Gigabit Midspan PoE
- 24v PoE Injector

- Industrial IoT
- IoT WiFi PLC
- Atomic Pi
- IoT Power Relay
- RS-232 Switch

- USB Loggers
- 16-Channel DIY
- T1/PRI Logger
- Personal Logger
- Accessories

- Logging Systems

- **Win 10 Appliance**
- **NG911 & Airband**
- **Military Radios**

Updated 11/24/2020

Advanced Power Control - Lua Scripting

On its own, a power switch isn't very smart. Add custom functionality using the built-in simple Lua scripting language. It's really simple. No programming experience is required. Give it a try!

Hardware Requirements

This page describes the Lua based scripting language used in DLI products with WiFi. If you're using a product without the WiFi option, look here for the BASIC scripting reference.

Entering Scripts

First, review this website for a quick overview of the Lua language. Log in as admin and use the scripting feature. Click the Scripting link on the left. Review these sample scripts to start:

Example - Turn lights (relays) on and off on a weekday schedule. Now includes an alternate event driven example.
Example - Switch an outlet on and off daily. Now includes an alternate event driven example.
Example - Turn on a sign during workdays - excluding holidays.

We have some additional sample custom scripts here.

```
--[[ This is a sample/test set of scripts for DLI controllers.

The scripting implementation has changed, and is no longer compatible
with the older BASIC implementation. The most important changes are:

- Now Lua-based.
- No more line numbers, blocks of code identified by functions.
- Most of ON, OFF, etc. are kept as legacy functions, which can be called like
 e.g  ON(2345), ON("2345") or ON "2345", your choice.

Execution is still based on threads. Now threads are more visible and
manageable. Try starting some and you'll see them appearing in the
list.
]]--

function turn_outlets_on()
    ON(1)
    ON(2)
    ON(3)
    ON(4)
    ON(5)
```

```
    ON(6)
    ON(7)
    ON(8)
end

function turn_outlets_off()
    OFF(12345678)
end

-- Hope this looks familiar so far.
-- Indentation is useful but not mandatory:
function toggle_stuff_and_log()
        LOG "One"
        ON(1)
        OFF(1)
        LOG "Two"
        ON(2)
        OFF(2)
        LOG "Done"
end

function do_some_lua_stuff()
        for i=1,8 do
                if i>4 then OFF(i) else ON(i) end
        end
end

function test_display()
        DISPLAY "\1Percent %%\v"
        DISPLAY "\2Backslash \\\v"

        -- These power displays only apply to the EPCR5 and EPCR6
        DISPLAY "\1%a\v" --  current Bus A
        DISPLAY "\2%A\v" --  voltage Bus A
        DISPLAY "\1%b\v" -- current Bus B
        DISPLAY "\2%B\v" -- voltage Bus B

        DISPLAY "\1%o\v" -- Outlets state in the form "12456" (ON are displayed)
        DISPLAY "\1%O\v" -- Outlets state in the form "++-+++--"

        DISPLAY "\1%n\v" -- Serial number
        DISPLAY "\2%f\v" -- Firmware version

        DISPLAY "\f"     -- Clear screen, first line intentionally blank
        DISPLAY "\2%d\v" -- System time/date

        DISPLAY "\1%M\v" -- MAC address of the power controller
        DISPLAY "\2%i\v" -- IP address of the power controller

        DISPLAY "\1%m\v" -- IP network mask
        DISPLAY "\2%g\v" -- IP gateway
end

--[[ Some additional scripts that may be of use. wait_until is used and required for many of these.
- The wait_until function is built-in, but shown here for reference and as an example
- local functions and variables must be declared before they are used
- local functions will not be displayed in the selection web UI box and cannot be used externally
- Firmware version  1.7.x introduced a new event system which can be used instead of the wait_until function and has more capabilities.
]]--

-- Swap the state of outlets 7 and 8
function flip_flop_7_8()
        if(outlet[7].state == on) then
                outlet[7].off()
                outlet[8].on()
        else
                outlet[7].on()
                outlet[8].off()
        end
end

-- Cycle an outlet daily at 1:00am
function cycle_outlet_daily()
        while true do
                wait_until({hour=1,min=0})
                outlet[2].cycle()
                delay(60) -- prevent it from running more than once in the same minute
        end
end

-- Cycle an outlet Sunday mornings at 2:00am
-- In the wait_until function, listed below, wday - the day of the week, Sunday = 1
function cycle_outlet_weekly()
        while true do
                wait_until({wday=1,hour=2,min=0})
                outlet[2].cycle()
```

```
                delay(60) -- prevent it from running more than once in the same minute
        end
end

-- toggle outlet 5 every 15 minutes past the hour
-- This example uses an event driven approach
function outlet_5_toggle_schedule()
  for i,t,data in event.stream(event.local_time({min=15})) do
    if outlet[5].state == on then
      outlet[5].off()
    else
      outlet[5].on()
    end
    log.notice("Outlet 5 was toggled")
  end
end

-- returns true if it's a weekend day
local function weekend(day_of_week)
        return day_of_week==7 or day_of_week==1
end

-- returns true if it's a weekday day
local function weekday(day_of_week)
        return day_of_week<7 and day_of_week>1
end

-- Turn on switches 1-5 on weekday mornings
function turn_on_lights_weekdays()
        while true do
                wait_until({wday=weekday,hour=8,min=0})
                for i=1,5 do
                        outlet[i].on()
                end
                delay(120) -- prevent it from running more than once in the same minute
        end
end

-- Turn off switches 1-5 on weekday evenings
function turn_off_lights_weekday_evenings()
        while true do
                wait_until({wday=weekday,hour=17,min=30})
                for i=1,5 do
                        outlet[i].off()
                end
                delay(120) -- prevent it from running more than once in the same minute
        end
end

-- Combine the minutes and hours to get total minutes
local function get_minutes(hours, minutes)
        return (hours*60)+minutes
end

-- Checks to see if a time is between two others
local function is__time_between(start_h, start_m, stop_h, stop_m, test_h, test_m)
        -- add 24 hours if endhours < starthours
        if (stop_h < start_h) then
                local stop_h_org=stop_h
                stop_h = stop_h + 24
                if (test_h <= stop_h_org) then -- if endhours has increased the current hour should also increase
                        test_h = test_h + 24
                end
        end
        -- The minutes within the day
        local start_t_val = get_minutes(start_h, start_m)
        local stop_t_val = get_minutes(stop_h, stop_m)
        local cur_t_val = get_minutes(test_h, test_m)
        return (cur_t_val >= start_t_val and cur_t_val < stop_t_val) -- cur_t_val < stop_t_val prevents including the last minute
end

-- Check to see if now is between start and end time
local function is_now_between(start_h,start_m,stop_h,stop_m)
        local time = os.date("*t")
        return is__time_between(start_h, start_m, stop_h, stop_m, time.hour, time.min)
end

-- Schedule the switch on between 8:30am and 5:15pm and monitor in between.
-- The limitation here is that this schedule cannot be overridden unless this script is stopped.
function schedule_switch_one()
        while true do
                if (is_now_between(8,30,17,15)) then
                        if(outlet[1].state == off) then
                                outlet[1].on()
                        end
                else
```

```
                        if(outlet[1].state == on) then
                                outlet[1].off()
                        end
                end
                delay(1) -- Don't be a CPU hog or it will get killed by the system
        end
end

-- Here we use wait_until to turn outlet 4 off at midnight.
function turn_off_light_at_midnight()
        wait_until({hour=0,min=0})
        OFF(4)
end

-- Here's an example function with loops. wait_until sleeps until a specified time
-- *** NOTE: You don't need to add this function to use it. It is built-in to the scripting server.
-- * wait_until parameters *
-- day - the day of the month, starting with 1
-- month - the month, January = 1
-- year - the year, with century
-- wday - the day of the week, Sunday = 1
-- yday - the day of the year, January 1 = 1
-- hour - the hour
-- min - the minute
-- sec - the second
-- ** It is not advised to perform exact matches on seconds since delays of internal
-- operations may be greater than 1 second.
-- ****************************
function wait_until(conditions)
        repeat
                local ok=true
                local date=os.date("*t")
                for k,v in pairs(conditions) do
                        if type(v)=="function" then
                                ok=v(date[k])
                        else
                                ok=date[k]==v
                        end
                        if not ok then
                                break
                        end
                end
                if not ok then
                        delay(1)
                end
        until ok
end

-- EPCR5 - EPCR7 only
-- Log some meter readings to the system log
function log_meter_readings()
        LOG("Old style of logging")
        LOG(string.format("Bus A voltage: %g, bus A current: %g",meter.buses[1].voltage,meter.buses[1].current))
        LOG(string.format("Bus B voltage: %g, bus B current: %g",meter.buses[2].voltage,meter.buses[2].current))

        log.info("This is the new format for logging")

        if(meter.buses[1].voltage > 95 and meter.buses[1].voltage < 130) then
                log.notice("Bus A voltage: %g, bus A current: %g",meter.buses[1].voltage,meter.buses[1].current)
        else
                log.warning("Warning! Bus A voltage: %g, bus A current: %g",meter.buses[1].voltage,meter.buses[1].current)
        end

        if(meter.buses[2].voltage > 100) then
                log.info("Bus B voltage: %g, bus B current: %g",meter.buses[2].voltage,meter.buses[2].current)
        else
                log.warning("Low Voltage Warning - Bus B voltage: %g, bus B current: %g",meter.buses[2].voltage,meter.buses[2].current)
        end

        if(meter.buses[2].voltage > 120) then
                log.notice("Bus B voltage: %g, bus B current: %g",meter.buses[2].voltage,meter.buses[2].current)
        else
                log.warning("Bus B voltage: %g, bus B current: %g",meter.buses[2].voltage,meter.buses[2].current)
        end
end

-- Reverse cycle - Cycle a switch on, then off
-- Remember that the "ON" speed between switches is limited by the "On sequence delay" in the Setup page
-- This function can be called from the autoping page and the switch(es) will be passed.
function reverse_cycle(selected_relays)
        count = #selected_relays
        for i = 1, count do
                outlet[selected_relays[i]].on()
        end
```

```
        delay(15)  -- Time "on" delay in seconds

        for i = 1, count do
                outlet[selected_relays[i]].off()
        end
end
```

Have a smart script or unique way to use your switch? Let us know!

engineering@digital-loggers.com